Controles técnicos de seguridad para la protección de aplicaciones web

La seguridad en las aplicaciones sigue siendo un aspecto clave en la denominada seguridad de la información. Múltiples puntos deben ser considerados en la protección de estas y variadas son las tecnologías y fabricantes que proponen soluciones. Pero ninguna de ellas es perfecta ni milagrosa. Es necesario considerar y valorar todo aquello que tenemos disponible para, con una unión casi nunca ideal, conseguir situar el nivel de seguridad allí donde en cada caso sea necesario, aplicando las dosis adecuadas de pragmatismo. Renovando y ampliando esa filosofía, la Fundación OWASP también es una fuente de propuestas y soluciones dignas de estudio cuidadoso y de una interesante consideración.

Vicente Aguilera Díaz / Daniel Fernández Bleda

Cualquier estadística reciente nos revela que las aplicaciones web son, a día de hoy, el componente más abusado para realizar una intrusión a nuestras infraestructuras telemáticas. Dado que el objetivo perseguido en la mayoría de ocasiones es el económico, no es de extrañar que los ataques se concentren en esta capa, ya que el negocio se encuentra en la web.

Las estadísticas de alguno de los sitios de referencia demuestran que los "clásicos" *defacements* aumentan, habiéndose "registrado" 1,5 millones en 2010; pero, lo más inquietante, es que se reconoce que los que han hecho de esto su trabajo ya no se limitan simplemente a modificar una página inicial, sino que instalan software para el robo de información en los mismos sitios cuya seguridad violan, no dando a conocer esos otros sitios vulnerados para no dañar su "trabajo" [1] por lo que los *sites 'troyanizados'* podrían ser un múltiplo desconocido de los publicados.

Si algunos análisis [2] sitúan entre 250 y 300 millones las páginas web en Internet, no hay estadísticas fiables de sitios sirviendo *malware* y las cifras varían entre el 0,5% y el 5% según lo que se considera *malware*. Lo más relevante, independientemente, es que el objetivo ya no es la información en los servidores que sirven las aplicaciones web, sino la de los usuarios de esas aplicaciones. Por tanto, ahora más que nunca, es imprescindible disponer de aplicaciones seguras. No nos podemos permitir desarrollar o acceder a otro tipo de aplicaciones sabiendo la responsabilidad que tenemos con los usuarios de estas.

Si queremos concienciar tendremos informes para todos los gustos dependiendo de las fuentes. Algunos nos mostrarán claramente que el robo de información financiera es uno de los principales objetivos de los atacantes, aunque es importante poner en contexto que, precisamente, esos informes han surgido a raíz de la investigación de brechas de seguridad y de incumplimiento normativo, principalmente de PCI DSS [3] [4]. Otros informes dejarán claro que las motivaciones de reconocimiento, superación de retos, políticas, etc., son las más importantes, sobre todo cuando las cifras surgen de "organizaciones" que incluyen rankings de atacantes.

Incluyendo el contexto en los diferentes análisis que se pueden consultar, hay algo claro y común a todos ellos, las brechas de seguridad explotadas se localizan en las aplicaciones y sus vulnerabilidades en cifras oscilan entre un 55 y 75 por ciento, habiendo en todos ellos un porcentaje de desconocimiento de la vía de explotación, que podría ampliar el rango al alza entre 5 y 15 puntos.

Según el renombrado Bruce Schneier [5], los profesionales de la seguridad ven el mundo de forma distinta al resto de personas. Evalúan constantemente su entorno y las decisiones que toman en función del riesgo de seguridad que determinan. Así, por ejemplo, no pueden entrar en una tienda sin analizar cómo podrían robar. Esta es la mentalidad de la que debería disponer cualquier persona involucrada en el proceso de desarrollo de aplicaciones. En lugar de pensar únicamente en el requerimiento funcional como objetivo,

deberíamos disponer de tiempo adicional para pensar en cómo puede ser atacada nuestra aplicación. En definitiva, pensar como un delincuente.

Llegado este punto, debemos ser conscientes de que la seguridad del software no equivale a escribir código de forma segura. Existen un gran número de actuaciones que deberíamos contemplar a lo largo del ciclo de vida del desarrollo de software para alcanzar el nivel de seguridad requerido por nuestra organización y para nuestra aplicación particular. Para conseguirlo, nos podemos apoyar en distintas metodologías y *frameworks* (*Security Development Lifecycle* de Microsoft, *Security Touchpoints* de Cigital, OpenSAMM de OWASP, etc.), aunque esto se escapa del objetivo de este artículo.

La presente exposición pretende abordar las tecnologías y controles técnicos que se pueden utilizar para incrementar el nivel de seguridad de los desarrollos, así como dar a conocer una taxonomía de las aplicaciones que permita definir el nivel de seguridad que se pretende alcanzar en cada caso.

Si partimos de la premisa de que resulta más fácil romper algo que diseñarlo para que no pueda ser roto, vemos que nos encontramos en una clara situación de desventaja. El atacante solo debe encontrar una deficiencia en nuestra aplicación para explotarla y conseguir así su objetivo, mientras que nosotros debemos protegernos de todas las amenazas existentes. No obstante, esto no debe servir de excusa para no implementar las medidas de seguridad necesarias, sino todo lo contrario, un estímulo para ganar esta partida.

TAXONOMÍA DE LAS APLICACIONES

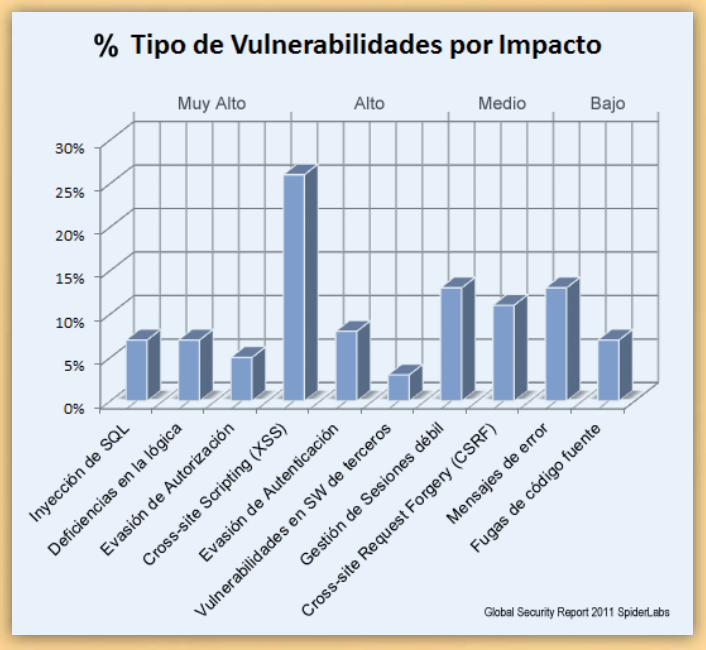
Resulta evidente pensar que no debemos aplicar los mismos controles de seguridad para todas las aplicaciones. El esfuerzo y el coste de la aplicación de estos controles no debería superar en ningún caso el valor del activo que se desea proteger. Por lo tanto, en primera instancia, nos vemos obligados a clasificar nuestras aplicaciones en distintos niveles, de forma que cada uno de los mismos requiera la adopción de unos determinados controles de seguridad.

En este sentido, OWASP (*Open Web Application Security Project* [6]) dispone del proyecto ASVS (*Application Security Verification Standard* [7]), que define cuatro niveles de confianza en la seguridad de las aplicaciones web, en función de la criticidad de las mismas. Es necesario entender el nivel de riesgo de nuestra aplicación para poder mapear este nivel con un nivel ASVS:

• **OWASP ASVS Nivel 1: Verificación Automatizada.** Según ASVS, este nivel es apropiado para aplicaciones donde únicamente se requiere cierto nivel de confianza en el correcto uso de los controles de seguridad. La verificación de la correcta implementación de los requerimientos de seguridad puede realizarse de forma automática, ayudándose de una revisión manual para descartar falsos positivos identificados por las herramientas.

• **OWASP ASVS Nivel 2: Verificación Manual.** Según ASVS, este nivel es apropiado para aplicaciones que manejan datos de carácter personal, realizan transacciones B2B, procesan información de tarjetas de crédito o procesan PII (*Personally Identifiable Information*).

• **OWASP ASVS Nivel 3: Verificación de Diseño.** Según ASVS, este nivel es apropiado para aplicaciones que realizan transacciones B2B significativas, incluyendo aquellas que procesan información médica, implementan funciones críticas de negocio o procesan otros activos sensibles.



• **OWASP ASVS Nivel 4: Verificación Interna.** Según ASVS, este nivel es apropiado para aplicaciones críticas, que protegen la vida y la seguridad de las personas, infraestructuras críticas o funciones de defensa. También se incluyen en este nivel aquellas aplicaciones que procesan activos sensibles.

OWASP ASVS no define únicamente los niveles en los que clasificar nuestras aplicaciones, sino que además identifica los requerimientos mínimos de seguridad que se deben verificar para catalogar nuestra aplicación en uno de los niveles ASVS. De hecho, ese es el principal cometido de este estándar.

MEDIDAS DE SEGURIDAD PARA LA PROTECCIÓN DE APLICACIONES WEB

Para conocer los controles técnicos y las tecnologías a utilizar para la protección de nuestras aplicaciones, hay que conocer previamente los requerimientos de seguridad que se desea alcanzar. Según *OWASP Development Guide* [8], estos requerimientos pueden agruparse en las siguientes áreas:

- Arquitectura de seguridad.
- Autenticación.
- Gestión de sesiones.
- Control de acceso.
- Validación de entradas.
- Codificación de salida y rutinas de escape.
- Criptografía.
- Gestión de errores y *logging*.
- Protección de datos.
- Seguridad en la comunicación.
- Seguridad en HTTP.
- Configuración de seguridad.
- Búsqueda de código malicioso.
- Seguridad interna.

El problema a la hora de implementar un mecanismo de protección reside en la ausencia de un control estándar. Nos enfrentamos a un amplio abanico de *frameworks* y librerías de seguridad (ACEGI, Struts, Anti-XSS, Jasypt, log4j, BouncyCastle, xml-dsig, JAAS, Java URL Encoder, etc.), donde cada uno de ellos proporciona solución a alguno de los requerimientos definidos en las áreas enumeradas anteriormente. Añadida a la propia tecnología para la integración de la seguridad en el desarrollo, dispondremos de tecnología externa a esta, como cortafuegos de aplicación y bases de datos o para la revisión activa de seguridad como herramientas de análisis de código estático y dinámico, siendo el mayor problema de todas ellas, su configuración y mantenimiento por la necesidad de conocimiento en la propia seguridad en aplicaciones.

OWASP cuenta con un proyecto que facilitará la tarea de desarrollar aplicaciones con un menor riesgo. Se trata del OWASP Enterprise Security API [9], más conocido como ESAPI, una librería que implementa un conjunto de controles de seguridad (colección de clases que encapsulan la mayoría de las operaciones de seguridad requeridas por las aplicaciones), que protegerán de fallos tanto en el diseño como en la implementación.

ESAPI cuenta con distintas versiones para ser utilizado por diferentes lenguajes (Java EE, .NET, ASP, PHP, ColdFusion/CFML o Python, entre otros), y todas se distribuyen bajo licencia BSD. Se permite el uso o modificación de ESAPI a nuestro antojo, y puede ser incluido en productos comerciales. Y todo esto, libre y de código abierto.

Procede ahora hacer un repaso de las áreas que debemos proteger en las aplicaciones (donde encontramos típicamente los principales requerimientos de seguridad) y las tecnologías que, como ESAPI, pueden ser utilizadas para implementar dichos requerimientos.

Arquitectura de Seguridad

Debemos entenderla como un conjunto de principios o máximas que debemos seguir durante el diseño de la aplicación. Para identificar las medidas de seguridad que se deberían

contemplar, los arquitectos deben pensar en cómo un usuario malicioso podría abusar de cada característica para ser utilizada con un fin distinto para el que fue diseñada.

Una arquitectura de seguridad debe proveer de los procesos y controles necesarios para garantizar la tríada clásica: confidencialidad (la información solo será accesible para aquellos usuarios autorizados a tener acceso), integridad (la información que se transmite no podrá ser alterada) y disponibilidad de los datos (la información siempre estará disponible cuando se requiera el acceso a la misma).

Entre los requerimientos de seguridad a este nivel, encontramos:

• Identificación de políticas de seguridad corporativas: será relevante considerar la utilización de marcos como CobIT e ISO 2700X; marcos regulatorios, ya que en función de los datos que maneje la aplicación estaremos obligados a cumplir determinadas normativas –como PCI DSS y PCI PA-DSS–; o la LOPD como más conocidas. Pero también habrá sectores con regulación propia que deberán tenerse en cuenta: financiero, sanitario, químico/farmacéutico, etc.

• Identificación de cada componente de la aplicación y la posible interacción entre los mismos, o la falta de perminso en la compartición de componentes por parte de la aplicación: por ejemplo, separar físicamente las capas de presentación, lógica y persistencia.

ESAPI WAF (*Web Application Firewall*) incorpora una capa adicional de seguridad que ayuda a minimizar los riesgos típicos. Basado en una política definida en XML, permite analizar el tráfico de entrada/salida de la aplicación y aplicar las reglas definidas por el usuario. La mayoría de vulnerabilidades identificadas por las herramientas automáticas pueden ser detectadas por ESAPI WAF impidiendo su explotación.

Existen en el mercado un gran número de herramientas que permitirán evaluar, de manera automática y en forma de caja negra, el nivel de seguridad de las aplicaciones e infraestructura relacionada. A nivel comercial, los escáneres más utilizados son: Acunetix, HP WebInspect e IBM Rational AppScan. Todos ellos son capaces de detectar un gran número de vulnerabilidades en función de la política de escaneo que se haya definido, y permitirán exportar los resultados a distintos formatos (útil para realizar un tratamiento posterior de esta información). Estas herramientas publican nuevas versiones con relativa frecuencia y se adaptan a las nuevas necesidades de las organizaciones, como la detección de incumplimientos normativos (como es el caso de PCI DSS [10]). No obstante, todas tienen un coste elevado que puede resultar prohibitivo para algunos (la mayoría). En caso de que nuestro presupuesto nos lo permita y puestos a elegir, ¿por cuál optar? Como acostumbra a ocurrir, cuando nos encontramos con algún dilema siempre hay alguien que tuvo que enfrentarse a él antes que nosotros. En este caso, puede consultarse el proyecto WAVSEP (*Web Application Vulnerability Scanner Evaluation Project* [11]), así como distintas comparativas existentes de herramientas [12] tanto comerciales como de código abierto.

En cambio, si el objetivo es identificar problemas de seguridad en nuestro código (se suele decir que si quieres saber qué es lo que hace realmente una aplicación, debes revisar su código), hay que conocer las alternativas de las que se dispone. En este caso, las herramientas de código abierto para análisis de código estático aún van un paso más atrás que en el caso de los escáneres de análisis dinámico de vulnerabilidades (donde allí sí pueden encontrarse herramientas *open-source* que ofrezcan resultados de cierta calidad). En la presentación [13] ofrecida por Luis Rodríguez Berzosa en el III OWASP Spain Chapter Meeting [14] encontramos una enumeración de herramientas comerciales y *open-source* con este fin.

Autenticación

El objetivo de este mecanismo es el de permitir identificar a los usuarios que acceden a la aplicación, de forma que no se permita el acceso indiscriminado y anónimo a la misma, además de facilitar la trazabilidad de los usuarios.

Existen un gran número de requerimientos que afectan a esta área. No olvidemos que es una de las más atacadas por resultar, en muchas ocasiones, la única interacción visible entre el usuario y la aplicación. A nivel general, este mecanismo debería implementar un control de acceso suficientemente robusto (adecuado al riesgo de la aplicación), asociar una entidad del sistema a un único usuario y denegar el acceso a todos aquellos usuarios que realicen ataques contra el sistema de autenticación.

OWASP ESAPI dispone de la clase "*Authenticator*", que incorpora un conjunto de funciones para generar y gestionar tanto las credenciales del usuario como los identificadores de sesión. De esta forma, se simplifica enormemente la labor del desarrollador a la hora de implementar las acciones derivadas del uso de este mecanismo (como iniciar/terminar una sesión, generar o modificar su contraseña, crear/eliminar usuarios u obtener los datos del usuario autenticado, entre otras funciones).

Podrán integrarse múltiples tecnologías para facilitar o simplificar la gestión de los procesos de autenticación; los más comunes serán la conexión a servicios del Directorio Activo de *MS Windows*, o servicios clásicos de LDAP. También desde hace años y, podría decirse que con olas (algunos les llaman modas), hemos pasado a emplear sistemas más abiertos como la Federación de Identidades y la *OpenID*, sobre todo empleados en entornos heterogéneos con la tendencia de desaparición de los clientes pesados y el uso masivo de aplicaciones web. A modo de anécdota, uno de los sectores que ha empleado estas tecnologías para simplificar el acceso a sus usuarios de forma simple ha sido el de las webs de descargas de contenidos protegidos. Hay quien lo muestra como ejemplo de su nivel de madurez cuando llega al usuario de esta forma tan masiva.

Gestión de sesiones

HTTP es un protocolo *stateless*, es decir, trata cada petición como si de una transacción distinta se tratara. De esta forma, es necesario implementar un mecanismo que permita mantener el estado de la conversación con un cliente. Es ahí donde entra en juego el sistema de gestión de sesiones.

La gestión de sesiones está, por lo tanto, íntimamente relacionada con la autenticación de usuarios, por lo que se trata de otro mecanismo crítico de la aplicación. La implementación de este mecanismo debe garantizar que la asociación entre el usuario y su sesión sea lo suficientemente robusta y criptográficamente segura. No hay que reinventar la rueda: actualmente cualquier *framework* incorpora su propia implementación de gestión de sesiones. Entre los requerimientos de esta gestión debe contemplarse la validación de los identificadores de sesión, el establecimiento de un tiempo de vida (las sesiones deben expirar tras un tiempo de inactividad o tras un tiempo prudencial), regenerar los identificadores (tras la autenticación de usuarios, tras un tiempo prudencial o al superar un número determinado de peticiones), destruir las sesiones de forma segura tras finalizar la sesión y asociar los identificadores de sesión con algún otro *token* criptográficamente seguro (por ejemplo, el certificado SSL del cliente) para autenticar a los usuarios.

OWASP ESAPI ayuda en la implementación de estos requerimientos de seguridad. Así, dentro de la clase "*Validator*" encontramos métodos que realizan la función de

validación de los identificadores de sesión, regeneración de dichos identificadores, o destrucción de la sesión de forma segura, entre otras acciones.

Control de acceso

Esta área cubre la necesidad de permitir el acceso a recursos privados únicamente a aquellos usuarios autenticados y que dispongan del nivel de privilegios adecuado. Este proceso, por lo tanto, se realiza tras la autenticación del usuario. Las consecuencias de una implementación defectuosa posibilitan, por ejemplo, el acceso a recursos no autorizados, alteración de la información o ejecución de operativas no autorizadas.

Este proceso debe garantizar que solo aquellos usuarios autorizados pueden llevar a cabo las acciones permitidas dentro del rol que se le haya asignado, acceder a recursos privados en función de su nivel de privilegios y prevenir la escalada de estos (por ejemplo, que un usuario con nivel de privilegios estándar no pueda ejecutar operativas de administración). Entre las técnicas para llevar a cabo este proceso, encontramos DAC (*Discretionary Access Control*), MAC (*Mandatory Access Control*) y RBAC (*Role Based Access Control*).

OWASP ESAPI dispone de la clase "*AccessController*", que define un conjunto de métodos para forzar el control de acceso a URLs, funciones de negocio, datos, servicios y ficheros. La interfaz de ESAPI permite centralizar la lógica del control de acceso de forma que simplifica a los desarrolladores el uso y verificación de este proceso.

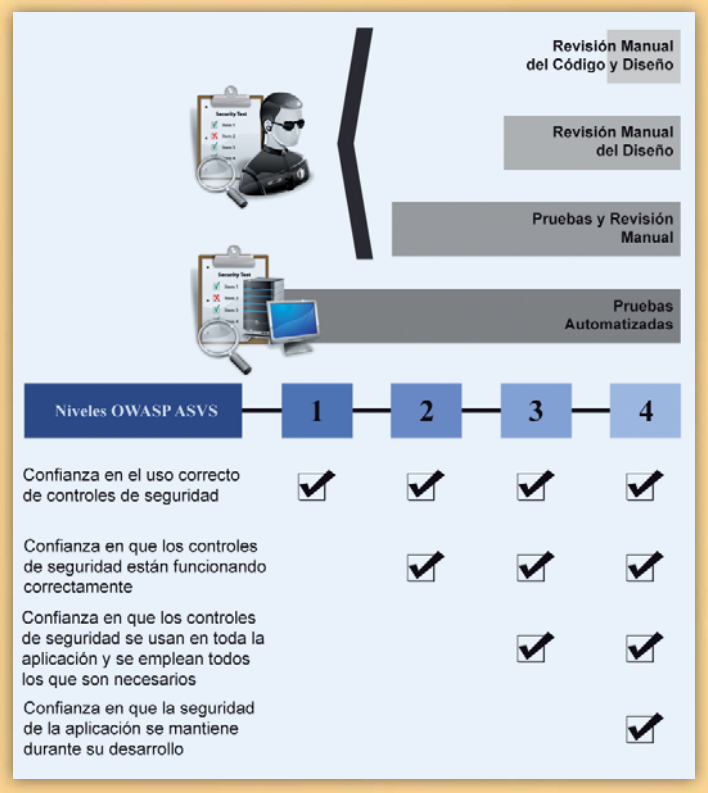
El control de acceso estará muy ligado a las soluciones de autenticación que hayamos desplegado. La tendencia de todas ellas ha sido simplificar, mediante la reutilización, los roles y permisos definidos en la empresa de forma centralizada. Precisamente, surgen un gran número de brechas de seguridad en la protección de la información a raíz de definiciones incorrectas de permisos por cambios no reflejados debidos a una dispersión en el alta, configuración y revocación de permisos.

Validación de entradas

Se trata del proceso de verificar que todas las entradas a la aplicación (ya sean provenientes del usuario, la infraestructura, entidades externas o bases de datos, entre otras fuentes) cumplen determinados criterios antes de ser utilizadas. Este es un proceso crítico, ya que la mayoría de riesgos en las aplicaciones se deben a deficiencias en la implementación de este mecanismo.

El proceso de validación de entradas debe implementarse en cada capa y dependiendo del contexto en el que nos encontramos: chequeos de integridad (*checksum*, HMAC, cifrado, firma digital), validaciones de datos (caracteres permitidos, sintaxis y longitud mínima/máxima) y validaciones en cuanto al cumplimiento de las reglas de negocio (por ejemplo, que un valor numérico se encuentre dentro de unos límites permitidos).

Existen distintas librerías que permiten incorporar, de forma segura, las validaciones de entradas evitando que los desarrolladores implementen este delicado proceso. Es el caso, por ejemplo, de ESAPI, que cuenta con la clase "*Validator*", en la que se incluyen un gran número de métodos para realizar estas validaciones. Además, los desarrolladores pueden extender las funcionalidades de esta clase para personalizar determinadas validaciones y/o adaptarlas a las características de sus aplicaciones.



Las tecnologías que han aportado (o lo han intentado) soluciones a la difícil tarea de validar las entradas han ido principalmente en el filtrado de las peticiones que alcanzaban las aplicaciones mediante cortafuegos de aplicación y, posteriormente, los de bases de datos. Pero si nos hacemos dos preguntas muy sencillas veremos que algo falla con las soluciones que los fabricantes aportan en esta área: ¿Quién emplea cortafuegos en su empresa? ¿Y cuántos cortafuegos de aplicación? Inconcebiblemente "los que respondan yo" a la segunda pregunta serán muchos menos (en varios órdenes de magnitud).

La razón del uso tan reducido de los cortafuegos de aplicación es simple y podemos llegar a la siguiente conclusión: si resulta que el mayor volumen de personal empleado en las TI incluye el desarrollo de software y, a una cierta distancia, el de sistemas, pero el nivel de seguridad de estos últimos es sensiblemente mayor –basándonos en los resultados de 10 años de historia de auditorías en las que las primeras siguen aportando una media del 80 por ciento de vulnerabilidades con un valor CVSS mayor de 7 (en una escala de 1 a 10)– ¿qué falla entonces? Generalizando, los primeros no tienen los conocimientos suficientes para implementar seguridad en el desarrollo de las aplicaciones y los segundos no los tienen para llevar a cabo una continuada gestión de estas soluciones.

Si añadimos a esto que una tecnología compleja dificulta su penetración por no ser comprendida de forma sencilla (cosa que queda fácilmente demostrada con la aparición de proyectos como el WAFEC (*Web Application Firewall Evaluation Criteria*), hace pensar que si todos los fabricantes de cortafuegos de aplicación se apuntaron a un proyecto para definir una metodología de evaluación de su producto para que el usuario decida cuál elegir en base a 10 conceptos "básicos" (arquitectura de despliegue, soporte HTTP y HTML, técnicas de detección, técnicas de protección, registro, *reporting*, gestión, rendimiento y XML/SOAP) debe ser por su complejidad. ¿Alguien se imagina respondiendo a casi 180 respuestas antes de elegir un antivirus, un cortafuegos de red o un anti-*spam*?

Algunos fabricantes han realizado esfuerzos en capacidades de *profiling* que consisten en implementar capacidades de aprendizaje y caracterización (principalmente) de los datos de los formularios de las aplicaciones web de forma automatizada. Otros, en integrar sus soluciones fuera de línea en el tráfico por la falsa convicción de muchos de sus clientes de que un cortafuegos de aplicación puede interrumpir un tráfico con mayor probabilidad que uno de red y con peores consecuencias. Pero la situación no ha cambiado: si presentamos a un miembro del operativo de gestión de un cortafuegos una traza de tráfico a un puerto no habitual, fácilmente lo podrá clasificar como un intento de ataque, pero si le presentamos una petición HTTP con una inyección de SQL codificada mediante una de las técnicas de codificación posible, ¿pondríamos la mano en el fuego en que la sabría identificar entre un tráfico legítimo? El resultado es que los fabricantes transmiten la incorrecta filosofía de que el *point & click configuration* es viable en sus productos, formando y certificando al personal en su gestión y no en la seguridad y los ataques que sufren las aplicaciones que protegen.

Añadido a esto hay que tener presente algunas consideraciones, desde el uso del DNI-e, y los sistemas de balanceo de carga externos, hasta el uso de protocolos, o bien obsoletos o bien demasiado recientes –que encapsulan código no HTTP y son reproduci-

dos por *plugins* en el navegador, etc.–, que, en algunos casos, sí pueden implicar los temidos cambios en las aplicaciones o implicar tráfico no completamente analizado.

Codificación de salida y rutinas de escape

La codificación de salida es el proceso por el que unos caracteres que pueden tener un significado especial (podrían ser interpretados como directivas o instrucciones) en un determinado contexto, sean convertidos a una cadena equivalente de texto sin posibilidad de ser interpretada de forma maliciosa. En determinadas aplicaciones, la codificación de salida no es un proceso trivial, ya que pueden existir múltiples contextos que analizar.

En este proceso, los desarrolladores deben conocer los contextos en los que se utilizarán los datos para aplicar la codificación de salida adecuada a cada uno de ellos. Cabe destacar que, para evitar que un usuario utilice múltiples codificaciones de forma maliciosa (por ejemplo, para evadir filtros de seguridad) en un parámetro de entrada, la aplicación debe "canonicalizar" (convertir el valor de entrada a su expresión más simple, decodificando todos los niveles) antes de realizar cualquier operación o cambio de contexto sobre dicha entrada.

OWASP ESAPI pone a nuestra disposición la clase "*Encoder*", que incorpora métodos para realizar las operaciones de codificación/decodificación de la salida/entrada, de forma que se realice de forma segura para distintos intérpretes. En su mayoría, toda la tecnología de cortafuegos de aplicación ha ido incluyendo tecnología para la decodificación del tráfico http, pudiendo evadirse los filtros de seguridad de algunos de ellos mediante técnicas de codificación.

Criptografía

El objetivo de este proceso es el de cifrar la información sensible (contraseñas, datos de tarjeta de crédito, datos médicos, datos personales, etc.) de forma que no pueda ser obtenida en claro por usuarios maliciosos aunque dispongan de acceso a la información cifrada. Entre los servicios para los que se requiere el uso de la criptografía, un caso típico es el de la autenticación (por ejemplo, mediante el uso de certificados digitales).

Desde el punto de vista de la seguridad, el requerimiento a seguir es el de garantizar que las operaciones de cifrado se realizan de forma segura para asegurar la confidencialidad e integridad de la información sensible utilizada por la aplicación.

OWASP ESAPI nos facilita la clase "*Encryptor*" para implementar las operaciones comunes de cifrado/descifrado, firma digital y funciones *hash*.

Gestión de errores y *logging*

Las aplicaciones se encuentran expuestas a distintos escenarios de error, producidos tanto de forma intencionada como casual. En caso de no gestionar adecuadamente estos escenarios, los mensajes de error de los componentes afectados pueden derivar en un alto riesgo para la aplicación (enumeración, divulgación de información sensible, *buffer attacks*, etc.). En caso de querer analizar una intrusión en los sistemas, la información almacenada en los *logs* resulta vital para que el resultado de la investigación forense permita identificar el origen del ataque y las deficiencias de las que se ha aprovechado.

Se dice que una aplicación bien desarrollada no conoce las distintas situaciones de error que pudieran producirse, pero tiene un plan para contemplarlas. Esto es más fácil de conseguir con una gestión de errores estructurada (frente a una gestión funcional), siempre que la tecnología de desarrollo lo permita. En caso de producirse una situación de error, la aplicación debe permanecer en un estado seguro, y no revelar más información que la estrictamente necesaria.

Asimismo, la aplicación debe almacenar información de todas aquellas actividades que afecten al estado del usuario u operativas sensibles, garantizar que la información no será sobrescrita o alterada (tanto por usuarios locales como remotos) y aplicar medidas para proteger los registros que se generen.

OWASP ESAPI incorpora la clase "*Logger*", que nos ayuda a realizar *logs* sobre eventos de seguridad. Logger define varios niveles de criticidad para los eventos (de mayor a menor gravedad: fatal, error, *warning*, info, *debug* y *trace*) y cuatro tipos predefinidos de eventos (SECURITY_SUCCESS, SECURITY_FAILURE, EVENT_SUCCESS y EVENT_FAILURE) que pueden ser modificados o ampliados por el desarrollador. De esta forma, se permite definir el tipo de evento que se desea almacenar, así como el nivel de criticidad para cada uno de ellos.

ASVS es el estándar a seguir si se realiza alguna de estas tareas de forma regular:

- Escaneos de vulnerabilidades
- Análisis estático de código
- Tests de penetración manual
- Revisión de código manual
- Modelado de amenazas
- Revisiones de la arquitectura de seguridad

Protección de datos

Esta área pretende agrupar aquellos requerimientos de seguridad cuyo fin es la protección de información sensible (números de tarjetas de crédito, números de pasaporte, PII, etc.).

Será necesario tener identificados todos los procesos que tratan datos sensibles y debe existir una política que, de forma explícita, defina cómo debe realizarse el acceso a estos datos y las medidas de seguridad que se deben aplicar (incluyendo el cifrado de esta información). Por otro lado, la aplicación debe evitar almacenar información sensible en el cliente y no transmitir datos sensibles en la URL. Una vez finalizado el tiempo requerido de retención de datos, debe existir un método para eliminar los datos sensibles que se hayan almacenado.

Nos podemos acercar a la protección de datos por varios frentes. Sin duda, el cifrado es el más empleado, más reciente es el uso de la *tokenización* y más desconocido (y que presenta más reticencias por las mismas comentadas anteriormente sobre los cortafuegos de aplicación) es el uso de cortafuegos de base de datos. Algunos de los fabricantes de cortafuegos de aplicación han desarrollado, aprovechando precisamente su conocimiento, sistemas particulares para bases de datos. Todos ellos deben considerarse por su increíble potencialidad en la protección de la información.

Seguridad en la comunicación

La práctica totalidad de las aplicaciones transmiten información sensible en algún momento de la comunicación entre el usuario y el servidor. Esta área define un conjunto de requerimientos que deben implementarse para garantizar que todas las comunicaciones con la aplicación se encuentran debidamente protegidas.

La aplicación debe asegurarse de que todas las conexiones autenticadas o que involucran funciones o datos sensibles, incluidas aquellas a otros componentes como los de *backend*, utilizan TLS (*Transport Layer Security*). Asimismo, todas las conexiones con sistemas externos en los que se transmiten datos sensibles deben ser autenticadas y utilizar un usuario con los mínimos privilegios necesarios para que la aplicación funcione como se espera.

Seguridad en HTTP

Este escenario agrupa un conjunto de requerimientos con el objetivo de 'securizar' la información que se transmite mediante el protocolo HTTP: peticiones y respuestas, sesiones, *cookies*, cabeceras, etc.

La aplicación debe garantizar que las redirecciones HTTP no incluyen datos no validados, que la aplicación acepta únicamente un conjunto predefinido de métodos HTTP (por ejemplo, GET y POST), cada respuesta HTTP especifica a través de la cabecera "*Content-Type*" un conjunto de caracteres seguro (por ejemplo, UTF-8), se utilizan atributos (Secure y HTTPOnly) para la protección de informa-

Decálogo de recomendaciones (10 consejos)

1. Defina distintos niveles de seguridad para sus aplicaciones y asigne, a cada uno de dichos niveles, unos requerimientos mínimos de seguridad que deban ser implementados.
2. Verifique periódicamente el cumplimiento de los requerimientos de seguridad definidos para sus aplicaciones.
3. Defina, diseñe, codifique y despliegue sus aplicaciones pensando más en el uso malicioso que alguien pueda realizar sobre las mismas, que en el uso esperado.
4. Huya de la solución "todo en uno". No podemos basar la seguridad de nuestras aplicaciones en un único producto o en el uso de una determinada tecnología.
5. Aplique medidas y controles de seguridad a distintos niveles. Siga el principio de defensa en profundidad.
6. Identifique los marcos regulatorios que afectan a sus aplicaciones. Defina políticas y procedimientos que le ayuden a su cumplimiento.
7. Otorgue a los requerimientos de seguridad la misma prioridad y consideración que a los requerimientos funcionales.
8. Manténgase informado sobre las amenazas existentes en las aplicaciones con el objeto de aplicar las medidas de seguridad que le ayuden a minimizar el nivel de riesgo.
9. Consulte las soluciones tecnológicas existentes en el mercado (sin menospreciar las *open-source*) y apóyese en especialistas del sector.
10. Potencie la cultura de la seguridad en su organización. La formación es un aspecto clave.

ción sensible en las *cookies*, valida que las cabeceras HTTP (tanto de peticiones como de respuesta) únicamente contengan caracteres imprimibles, y se genera un *token* criptográficamente seguro asociado al identificador de sesión para todas aquellas peticiones que realizan operaciones sensibles.

Configuración de seguridad

Como configuración de seguridad debemos entender todos aquellos requerimientos que han de implementarse para garantizar el almacenamiento seguro de la configuración sobre las características de seguridad de la aplicación.

De esta forma, la aplicación debe garantizar que todos aquellos aspectos de configuración relacionados con la seguridad disponen de un robusto control de acceso, que cualquier cambio en las características de seguridad queda reflejado en los *logs* de la aplicación y que no se permite el acceso a la misma si no es posible acceder a la información sobre la configuración de seguridad.

Búsqueda de código malicioso

La aplicación debe disponer de mecanismos que permitan la detección de código malicioso y la verificación de la integridad del código (librerías, ejecutables, ficheros de configuración, etc.) mediante *checksums* o *hashes*.

La tecnología disponible para cubrir este aspecto es realmente incipiente y en evolución. Tenemos el acercamiento de las casas de productos anti-*malware* de forma "pasiva": detectar *malware* en las aplicaciones web basado en patrones, de forma limitada en comportamiento o, insuficientemente, en servicios de listas negras de webs a las que se consulta a cada *website* navegado. Existe otro acercamiento, costoso en recursos pero con una efectividad mucho mayor, que consiste en el uso de *honeyclients*, *sandboxes* de navegadores y análisis dinámico de código [15].

Seguridad interna

Esta parcela se compone de aquellos requerimientos de seguridad que permiten que la propia aplicación incorpore métodos para protegerse ante fallos en la codificación.

De esta forma, la aplicación debe garantizar la protección de los atributos y políticas utilizadas por los controles de acceso por parte de accesos no autorizados, que los controles de seguridad son lo suficientemente simples de utilizar por parte de los desarrolladores y que la aplicación protege variables y recursos compartidos de accesos concurrentes.

Como hemos visto, los aspectos clave en el desarrollo de las aplicaciones implican considerar numerosos requerimientos de seguridad en distintos contextos. La responsabilidad, por lo tanto, no recae en los desarrolladores únicamente, y cada rol del equipo debe conocer y entender los requerimientos de seguridad que se les exige.

CONCLUSIONES

Las brechas de seguridad que se explotan en la actualidad se aprovechan, en su gran mayoría, de deficiencias en la capa de aplicación. A fecha de hoy, seguimos viendo aplicaciones que sufren vulnerabilidades que se conocen desde hace una década. Está claro que, o nuestra ignorancia provoca el desconocimiento de las amenazas existentes o no hemos comprendido el problema, su dimensión o su solución.

Como expresa el recurrido Sun Tsu en "El arte de la guerra", debes conocer a tu enemigo. Debemos ponernos en la piel del delincuente y conocer sus técnicas para poder aplicar medidas de seguridad con ciertas garantías. En esta guerra no estamos solos, y contamos con el conocimiento y tecnologías suficientes para alcanzar el nivel de seguridad requerido por nuestras aplicaciones. Y esto no siempre implica un desembolso importante.

No debemos olvidar que la seguridad es una medida más del nivel de calidad de la aplicación, y que involucra a todos los roles implicados en el ciclo de vida de las aplicaciones. Cada uno de ellos debe conocer y entender sus responsabilidades desde el punto de vista de la seguridad. ■

Vicente Aguilera Díaz
Spain Chapter Leader
OWASP
CISA, CISSP, CSSLP, ITIL
CEH|I, ECSP|I, OPSA, OPST
vicente.aguilera@owasp.org

Daniel Fernández Bleda
Director Comercial
INTERNET SECURITY AUDITORS
CISM, CISA, CISSP, ISO27001LA
OPSA, OPST, CHFI
dfernandez@isecauditors.com

Referencias

[1] http://www.zone-h.org/news/id/4737
[2] http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers
[3] http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf
[4] https://www.trustwave.com/downloads/Trustwave_WP_Global_Security_Report_2011.pdf
[5] http://www.isecauditors.com/es/des-prensa.html
[6] http://www.isecauditors.com/downloads/prensa/SIC-83_Servicio_Vigilancia_Anti_Malware.pdf
[7] http://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html
[8] http://www.owasp.org
[9] http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
[10] http://www.owasp.org/index.php/OWASP_Guide_Project
[11] http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
[12] https://www.pcisecuritystandards.org
[13] http://code.google.com/p/wavsep/
[14] http://www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf
[15] https://www.owasp.org/images/d/da/OWASP_Spain_20080314_Herramientas_de_análisis_estático_de_seguridad_del_código_estado_del_arte.pdf